Seascape Developer's Reference v3.2

Segmentation and Cover Classification Analyses of Seabed Images

Please use the following citation in published literature using Seascape:

Teixidó N, Albajes-Eizagirre A, Bolbo D, Le Hir E, Demestre M, Garrabou J, Guigues L, Gili JM, Piera J, Prelot T, Soria-Frisch A (2011) High-resolution software for Segmentation and Cover Classification Analyses of Seabed Images (Seascape). Mar Ecol Prog Ser 431: 45-53

Seascape is an open-source platform and the compiled software, source code, developer guide, and user manual are available online at this site (http://www.seascapesoft.com/documentation). Seascape is available as free software under the terms of the Free Software Foundation's (http://www.gnu.org/) GNU General Public License (GPLv3) (http://www.gnu.org/licenses/gpl.html) in source code form.

Table of contents

Brief Specification version 3.2

This application has been developed with wx Widgets library. In the development package the source code in C++ is provided, together with the working solution for Visual Studio 2003.

The code is compounded by the **Sxs** and **Lgl** libraries for image segmentation, the wx widgets library, the classes implementing, the interface (explained below) and the auxiliary libraries *BasicExcel* and *Tinyxml*.

InterfaceMedSeg is the main class and the entry point of the application. This is the class the wx engine will call at the startup of the application (as stated on sxs.cpp). This is the class containing pointers to all other classes needed by the application and the class modeling the GUI interface. It contains a member of the class **LeftWindow** for the left frame of the application. It contains a **ScrollWinMedSeg** member for the frame of the application where the image will be drawn.

InterfacecMedSeg also contains members for **ManageClassMedSeg**, **ManageColorMaps** and **AddClassMedSeg** classes which will inherit from window class of wx library and will contain the management pop up windows for species, colormaps, etc.

InterfacecMedSeg also contains members for **languageloader** and **locationloader** classes which will manage language strings and species data. Each time the application needs to retrieve or update any data regarding language strings or species items, it will be done through function calls to those members.

ImageMapping class is used to contain information about the image to be segmented, and **LayerOnSelection** class is used to manage the layers to be drawn over the image to be segmented. The layers will contain the colored regions for each classified species.

Sxs and lgl classes are the classes used by the classification engine (the **SxS** and LGL **libraries**). *BasicExcel* and *TinyXML* are classes from the two libraries used to work with *MS Excel* format and *XML* format.

Optimizations made on the segmentation algorithm and on the code

In this version, the underlying segmentation library has been optimized to enable the possibility of working with Seascape processing large resolution images. To achieve such optimization, two aspects of the segmenting algorithm have been upgraded and one new parameter to the interface has been added. A new dialog window has been added to the library in order to require the user to input some parameters before the segmentation process is run.

The two upgrades on the segmenting algorithm (and, thus, the library) are: The variability added to the node energies complexity and the variability on the criteria and threshold used for the base segmentation of the first stage of the algorithm.

Brief description of the segmentation algorithm

The segmentation algorithm is compounded on three major stages: Base segmentation, graph construction and scale climbing. The implementation of the algorithm is specified on the file *sxsImageScaleClimbing.cpp* of the SxS library.

Base segmentation

In this stage, the original image is segmented in different regions following a criterion. The input will be the original image, and the output is a set of different regions. Each of these regions will be compounded by the pixels, geographically adjacent, which the difference among them regarding the segmentation criterion is lower than a fixed threshold. Therefore, the lower this threshold is fixed, the more different resulting regions will be obtained, being the pixels inside each set more similar among them.
This stage is implemented on the function *computeBaseSegmentation()*.

Graph construction

In this stage, the input is the set of different regions resulting from the previous stage, and the output is a graph. This graph is build by creating a node per each of the resulting sets from the previous stage and creating a vertex between two nodes if the two nodes are geographically adjacent. Moreover, in this stage an energy value is computed for each vertex. This value will be set with regards to the difference between the nodes being connected. The way this energy value is computed will affect on the computational cost of the algorithm. The more complex this energy is, the higher the computational cost will be.
This stage is implemented on the function *buildGraph()* .

Scale climbing

In this stage, the input will be the enriched graph resulting from the previous stage, and the output is the whole structure resulting from the segmentation algorithm. From the enriched graph, a tree is build. First step is to create a branch on the tree from each of the nodes in the graph. Such nodes will keep all the information contained on the graph, including the vertex with the energies. Therefore, this step just involves a change of representation of the data. The important step is the next one, the scale climbing. This is an iterative process by means of which, at each interation, the pair of nodes with the lowest coupling energy are attached to a new node that will be parent of the two on the tree. For the new node all the energies are recomputed, obtaining a new coupling energy between the new node and all the other remaining nodes on the tree.
Repeating such iteration, a binary tree is obtained, being the node created on the last iteration the root node of the tree. One of the more important aspects of this algorithm is the fact that, during the iterative process, a structure is build, so at the time of using the tree, it results easy to find the height on the tree representing a determined segmentation scale value.
This stage is implemented on the functions *buildBase()*, *buildHeap()*, *climb()* and *postProcess()*.

Energies complexity

To establish a variable level of energy complexity, instead of a fixed level as before, we had to modify the class '**PiecewiseAffineFunction**' on the ' **sxsPiecewiseAffineFunction.h**' of the SxS library. We added a private member of the class of type '**unsigned int**' and labeled '**maxNPieces**', as well as modified the function that returns that value. We also modified the constructor of the class, so each new instantiation had to be created with a set value for this member. In order to maintain the consistency of the library, we had to modify the classes '**ImageScaleClimbingParameters**' and '**ScaleClimbingParameters**' in order to contain this new parameter. Also, in order to make the library use this new parameter, we had to modify '**ImageScaleClimbing**' and '**ScaleClimbing**' classes.

In order to make the interface use this new feature of the library, we added on the new dialog window a slider that the user will use to select a value for the level of complexity. This value will be send to the segmentation library though the '**ImageScaleClimbingParameters**' class.

Base segmentation criteria and threshold

Two aspects were upgraded regarding the base segmentation stage of the algorithm: The criteria used to perform such base segmentation and the threshold value used for it. The criteria were modified due to the existence of a bug on the library that implied the usage of only one channel of information for the base segmentation. Moreover, a new function coding has been added, so it is possible to add new different criteria on future versions of the library. The implemented one has been the Euclidean distance between pixel channel values in order to decide if differently segment two pixels or not. The other aspect is the addition of variability to the threshold of segmentation. That is, adding a variable to modify on execution time the behavior. In this release, this threshold is applied to the current segmentation criteria. That is, the parameter selected will affect the euclidean distance value needed to differently segment two pixels.

To implement this upgrades we modified the '**ConnectedComponents**' function on '**lglImageBasicAlgorithms_code.h**' file. On the same file we introduced the new function '**connectedCritera**' that implements the Euclidean distance criteria. This late function is now used by '**ConnectedComponents**' solving at the same time the previous bug of the library.

Again, modifications on '**ImageScaleClimbingParameters**' and '**ScaleClimbingParameters**' classes were necessary to keep libraries consistency. Also a slider has been implemented on the new dialog window in order to require the user to enter the value for such threshold.

Selection of the number of scales to work with

One of the main memory consuming parts of the application is the creation of several bitmap images on memory in order to work with different levels of scales. Three bitmaps files of the size of the original image times the selected zoom level are needed for each desired level of scale to work with. Therefore, it is important to be able to select how many scale levels the user needs. To require the user such parameter, a slider has been added on the new dialog window. This will set the value of a variable labeled '**m_nlevels**' that will substitute the compiling time defined constant, previously labeled 'NB_ECHANT_SEG'.

Specification of the new classes implemented on version 3.2

Segmentation Parameters Window

This is the class implementing (through inheritance of wxFrame) the new window requiring the user the parameter values for the segmentation algorithm. It contains three sliders from the wxWidgets library. It is worth noticing that the actual submission of the parameters to the segmentation function is done at the class' function OnClose (thrown while the user closes the window). This function creates a ImageScaleClimbingParameters instantiation with the gathered values from the sliders. Then, with such instantiation, it destroys itself (as instantiation of wxFrame) and then it throws the DoExecution function from the interface in order to perform the segmentation.

```
class SegmentationParametersWindow : public wxFrame
  wxSlider *m_slider_baseSeg_thrd;
  wxSlider *m_slider_energyComplexity;
  wxSlider *m_slider_levels;
  void OnClose(wxCloseEvent& event);
```

Table Window

This is the class implementing (through inheritance of wxFrame) the new window for displaying the current state of the classified polygons table. It will contain an wxGrid, and the constructor will have an integer parameter *higlight*. If *highlight* is set at the time of creating the instantiation, then the corresponding row of the table will be highlighted. This way, the same class can be used for normally displaying the table as well as for displaying the table with highlighting one one of the polygons.

```
class TableWindow : public wxFrame
  wxGrid *tableGrid;
  TableWindow(wxWindow *parent, wxWindowID id, const wxString& title, int highlight);
```

Segment Sizing Window

This is the class implementing (through inheritance of wxFrame) the new window that will require the user to input the size of the segmented. It is worth noticing that on the function *OnClose*, the class will update the values for the sizing of the image.

```
class SegmentSizingWindow : public wxFrame
  void OnClose(wxCloseEvent &ev);
```

Navigation Window

This is the class implementing (through inheritance of wxFrame) the window that will display a miniature of the image being processed and a frame over it corresponding to the current zoom view set on the interface. This window will be floating on top of the application while activated. It is important that this class has to implement wrapper functions for the paint and repaint events on the wxWidgets' loop library. Being closs-platforming a key issue on this release, such wrapping functions have been developed to correctly perform on any host OS.

```
class NavigationWindow : public wxFrame
  bool m_flagRefresh;
  wxString *_imagePath;
  void OnPaint(wxPaintEvent &ev);
  void RePaint(wxScrollWinEvent &ev);
```

Export XLS Window

This is the class implementing (through inheritance of wxFrame) the new window that will require the user to input variable fields to attach to the exported XLS file.

```
class ExportXLSWindow : public wxFrame
```

My About Window

This is the class implementing (through inheritance of wxFrame) the new window that will display the credits of the application, as well as the logos of the involved institutions.

```
class MyAboutWindow : public wxFrame
```

wx Image Panel

This is a custom widget class developed (from http://wiki.wxwidgets.org/An_image_panel) to easily display images on the application used to set many different small images for the logos on the about window.

```
class wxImagePanel : public wxPanel
 void paintEvent(wxPaintEvent & evt);
 void paintNow();
 void OnSize(wxSizeEvent& event);
 void render(wxDC& dc);
```

UML diagrams for interface classes

Attached to this document there are provided two UML diagram picture files. First diagram displays the main classes involved in the interface, and the second diagram displays all the classes involved in the interface.

Implementation of polygon routines. How to extract polygonal data from raster data

To obtain polygon data, such as area and perimeter, from raster data (such as a bitmap, our data), some routines to transform the data were implemented.
These routines attain two steps:

- **Extracting isolate regions from a bitmap set**
- **Computing area and perimeter values of the polygons formed by the isolated regions.**

This process is computationally expensive with large sets of pixels, and it is run for a region every time a new pixel is added to a region. Therefore, if the user is labeling a very large region with the same species, a slowdown of the interaction speed will be noticed.

Extracting isolate regions from a bitmap set

For the first stage, the method of fill flooding **Error! Reference source not found.** was implemented. With an initial set of pixels, we start from the first one of the set, and run a fill flooding procedure, separating from the initial set the pixels reached by the flooding and keeping the remaining pixels for the next iteration. Once the initial set is empty, we obtained as many sets as isolated regions there are on the initial set.

Computing area and perimeter values of the polygons formed by the isolated regions

Once we have the pixels of each region isolated, we needed to obtain from the pixels the area and the perimeter of the corresponding polygon for the region. The area is computed by determining the side length of the pixels, and then simply obtaining the area of each pixel times the number of pixels in the set.

For the perimeter computation of the polygon, we implemented a methodology presented in **Error! Reference source not found.** that apparently attains good results. The idea is obtain the contribution of each pixel on the set to the perimeter. This is

achieved by classifying each pixel on one of the 256 different classes of pixel established (each class is for each possible configuration of pixel depending on how many and where contiguous neighbors the pixel has). If a pixel has 8 neighbors, this makes 256 possible combinations. Each of the 256 classes has a contribution value. Each pixel is classified according to its neighbors, obtained the contributing value and added this last to the total perimeter of the polygon.

References

[1] Flood filling. http://en.wikipedia.org/wiki/Flood_fill

[2] Prashker, S. 'An Improved Algorithm for Calculating the Perimeter and Area of Raster Polygons', Geocomputation '99, Fredericksburg, Virginia, July 24, 1999. http://www.geovista.psu.edu/geocomp/geocomp99/Gc99/076/gc_076.htm